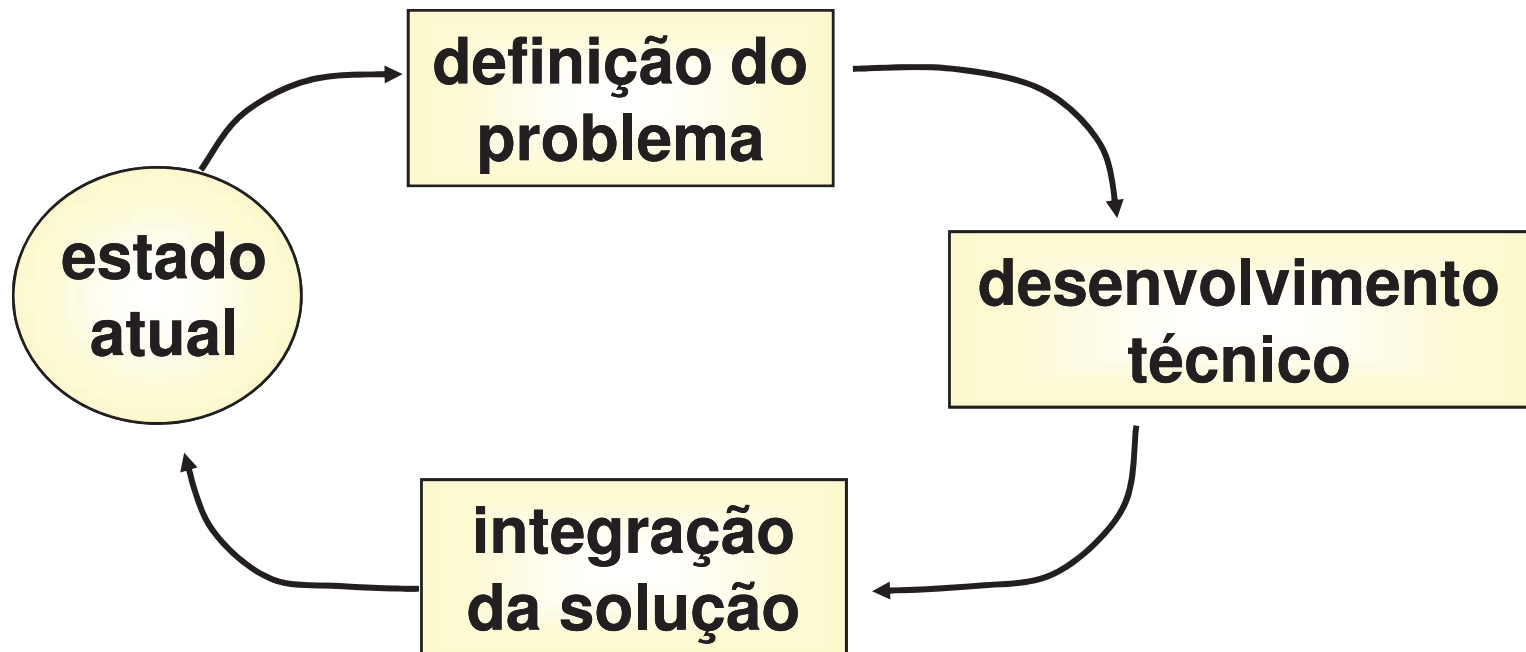


# Modelo de Processo de Software

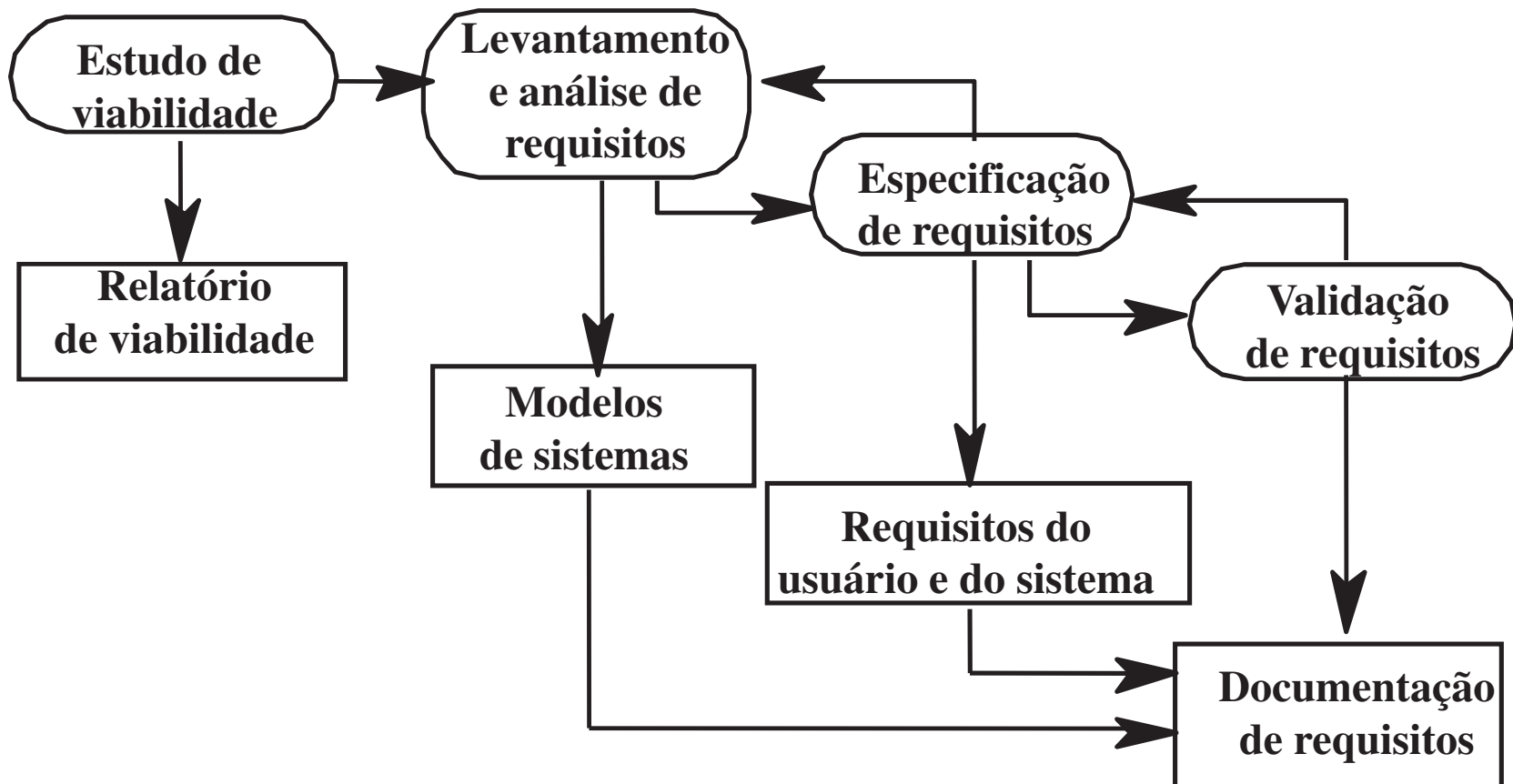
---

**Processo** → devem incorporar uma *estratégia* de desenvolvimento



# Modelo de Processo de Software

---



# Modelo de Processo de Software (paradigmas)

---

- **Modelo Sequencial Linear** (ciclo de vida clássico)
- **Modelos Evolucionários**
  - **Prototipação**
  - **Incremental**
  - **Espiral**
  - **Métodos Ágeis**
- **Modelo de Métodos Formais**
- **Técnicas de 4a Geração**

# Modelo de Processo de Software (paradigmas)

---

**É escolhido com base:**

- **na natureza do projeto e da aplicação.**
- **nos métodos e ferramentas a serem utilizados.**
- **nos controles e produtos que precisam ser entregues.**

# Modelo Sequencial Linear

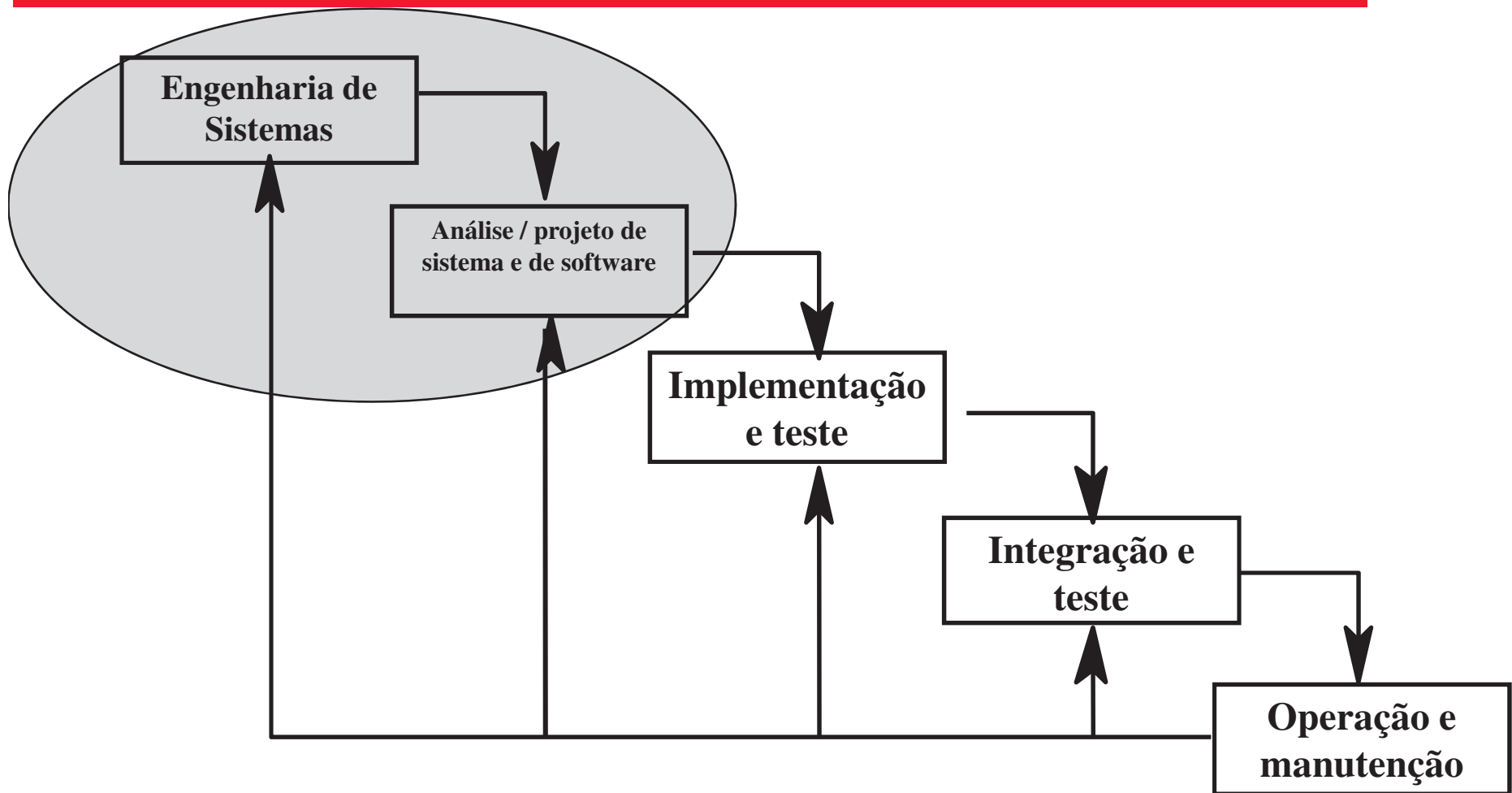
## (ciclo de vida clássico)

---

- Método sistemático e sequencial
- O resultado de uma fase se constitui na entrada da outra.
- Cada fase é estruturada como um conjunto de atividades que podem ser executadas por pessoas diferentes, simultaneamente.

# Modelo Sequencial Linear (ciclo de vida clássico)

---



# Modelo Sequencial Linear (ciclo de vida clássico)

---

- **ENGENHARIA DE SISTEMAS**

- envolve a coleta de requisitos em nível do sistema, pequena quantidade de projeto e análise de alto nível;
- deve-se **analisar os requisitos**, recursos e restrições para:
  - **apresentar soluções;**
  - **estudar a viabilidade;**
  - **planejar e gerenciar** o desenvolvimento a partir de **estimativas e análise de riscos** que se utilizam de **métricas**

**Definir quais os requisitos do produto de software, sem especificar como esses requisitos serão obtidos.**

# Modelo Sequencial Linear (ciclo de vida clássico)

---

- **ANÁLISE DE REQUISITOS DE SOFTWARE**

- processo de coleta dos requisitos é intensificado e concentrado especificamente no software.
- deve-se compreender o domínio da informação, a função, desempenho e interfaces exigidos.
- os requisitos (para o sistema e para o software) são documentados e revistos com o cliente.

**Resultado** → o contrato de desenvolvimento



# Modelo Sequencial Linear

## (ciclo de vida clássico)

---

- **PROJETO**
  - É definida a solução do problema
  - **Concentra-se em:**
    - Estrutura de Dados;
    - Arquitetura de Software;
    - Detalhes Procedimentais e
    - Caracterização de Interfaces.

**Resultado** → documentação de especificação de projeto

# Modelo Sequencial Linear

## (ciclo de vida clássico)

---

- **IMPLEMENTAÇÃO**
  - tradução das representações do projeto para uma linguagem “artificial” resultando em instruções executáveis pelo computador.
  - O projeto é transformado em um programa, ou unidades de programa. ( Teste de Unicidade)
  - **Resultado** → coleção de programas implementados e testados.

# Modelo Sequencial Linear (ciclo de vida clássico)

---

## • INTEGRAÇÃO

- Programas ou unidades de programas são integrados e testados como sistema.
- Integração incremental → programas ou unidades são integradas à medida em que forem sendo desenvolvidos.

**Resultado** → produto pronto para ser entregue ao cliente.

# Modelo Sequencial Linear

## (ciclo de vida clássico)

---

- **OPERAÇÃO / MANUTENÇÃO**

- **Operação**

- Instalação e configuração
- Utilização – inicialmente operado por um grupo de usuário

- **Manutenção**

- Corretiva: correção de erros remanescentes
- Adaptativa: adaptação dos produtos às mudanças novas versões e novas situações de operação – hardware, sistemas operacionais.
- Evolutiva: alteração dos requisitos e manutenção da qualidade.

**Resultado** → produto em funcionamento.

# Problemas com o Modelo Sequencial Linear

---

- Rigidez. Qualquer desvio é desencorajado
- Todo o planejamento é orientado para a entrega do produto de software em uma data única.
- Processo de desenvolvimento pode ser longo e a aplicação pode ser entregue quando as necessidades do usuário já tiverem sido alteradas.
- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe.
- Logo no início é difícil estabelecer explicitamente todos os requisitos. No começo dos projetos sempre existe uma incerteza natural.

# Modelo Sequencial Linear

## (ciclo de vida clássico)

ETAPA	PERGUNTAS-CHAVES	CRITÉRIOS DE SAÍDA
Definição do problema	Qual é o problema	Declaração da delimitação e objetivos.
Estudo de viabilidade	Há uma solução viável	Análise geral de custo/benefício Alcance e objetivos do sistema.
Análise	O que terá de ser feito para resolver o problema?	Modelo lógico do sistema: Diagrama de Fluxo de Dados; Diagrama de Entidade e Relacionamento Diagrama de Transição de Estado; Dicionário de Dados; Especificação de Processos. UML.

# Modelo Seqüencial Linear

## (ciclo de vida clássico)

ETAPA	PERGUNTAS-CHAVES	CRITÉRIOS DE SAÍDA
Projeto	Como o problema deve ser resolvido? Como o sistema deve ser implementado?	Soluções Alternativas Especificação de hard/soft; Plano de implementação; Plano de teste preliminares; Procedimento de segurança; Procedimento de auditoria.
Implementação	Faça	Programas; Plano de testes; Procedimento de segurança; Procedimento de auditoria.
Teste	Verificar o sistema	Testes do geral do sistema.
Manutenção	Modificar o sistema conforme necessidade.	Apoio continuado.

# Modelo Evolucionário

---

- Abordagem baseada na idéia de desenvolver uma implementação inicial, expor o resultado ao comentário do usuário e fazer seu aprimoramento por meio de muitas versões.
- As atividades de desenvolvimento e validação são desempenhadas paralelamente, com um rápido feedback entre elas.
- Os detalhes e extensões ainda devem ser definidos

**Ex: editor de texto**



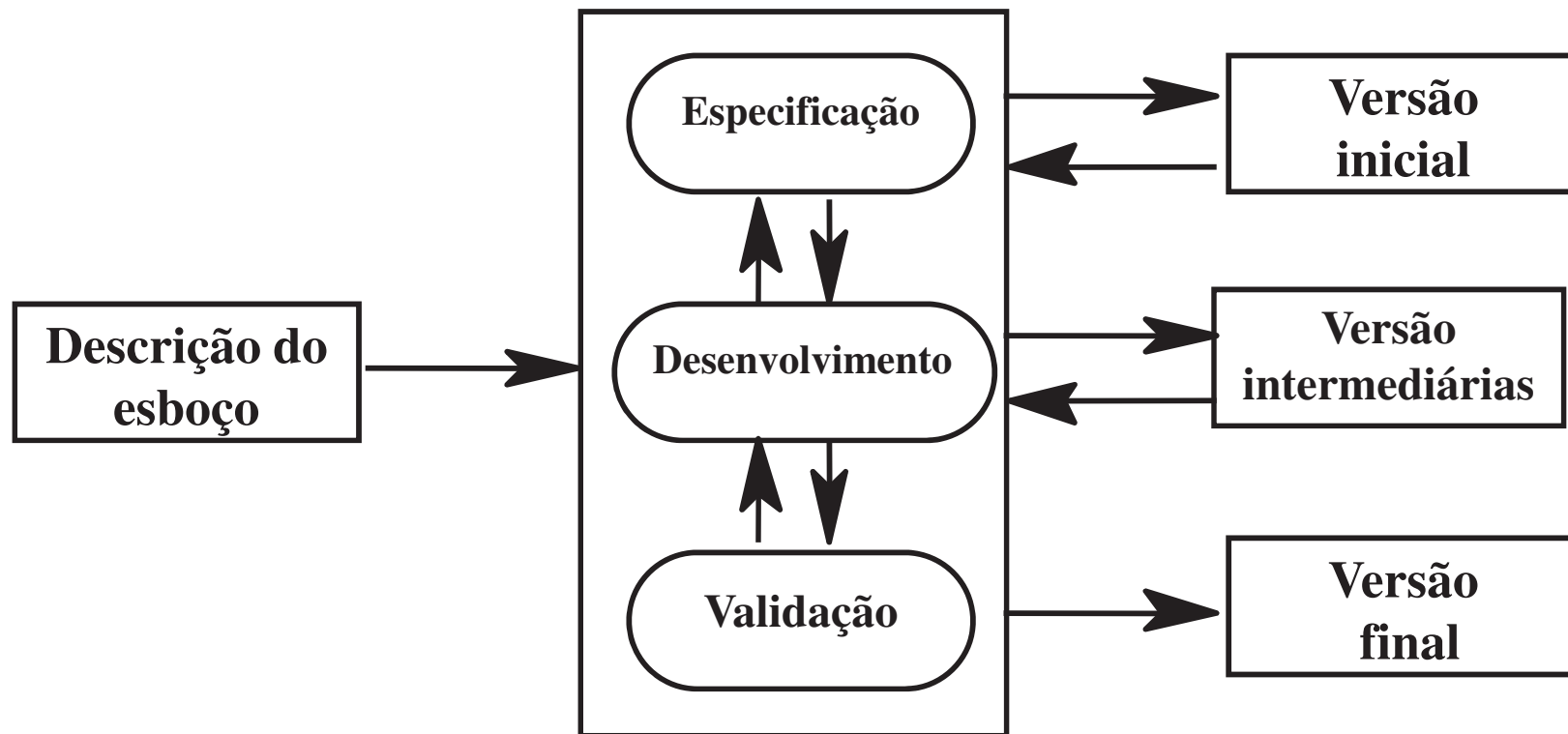
# Modelo Evolucionário

---

- Trabalhar junto com o cliente, a fim de explorar seus requisitos e entregar um sistema final.
- O desenvolvimento se inicia com as partes do sistema que são mais bem compreendidas.
- O sistema evolui com o acréscimo de novas características à medida que elas são propostas pelo cliente.
- Importante quando é difícil, ou mesmo impossível, estabelecer uma especificação detalhada dos requisitos do sistema a priori.

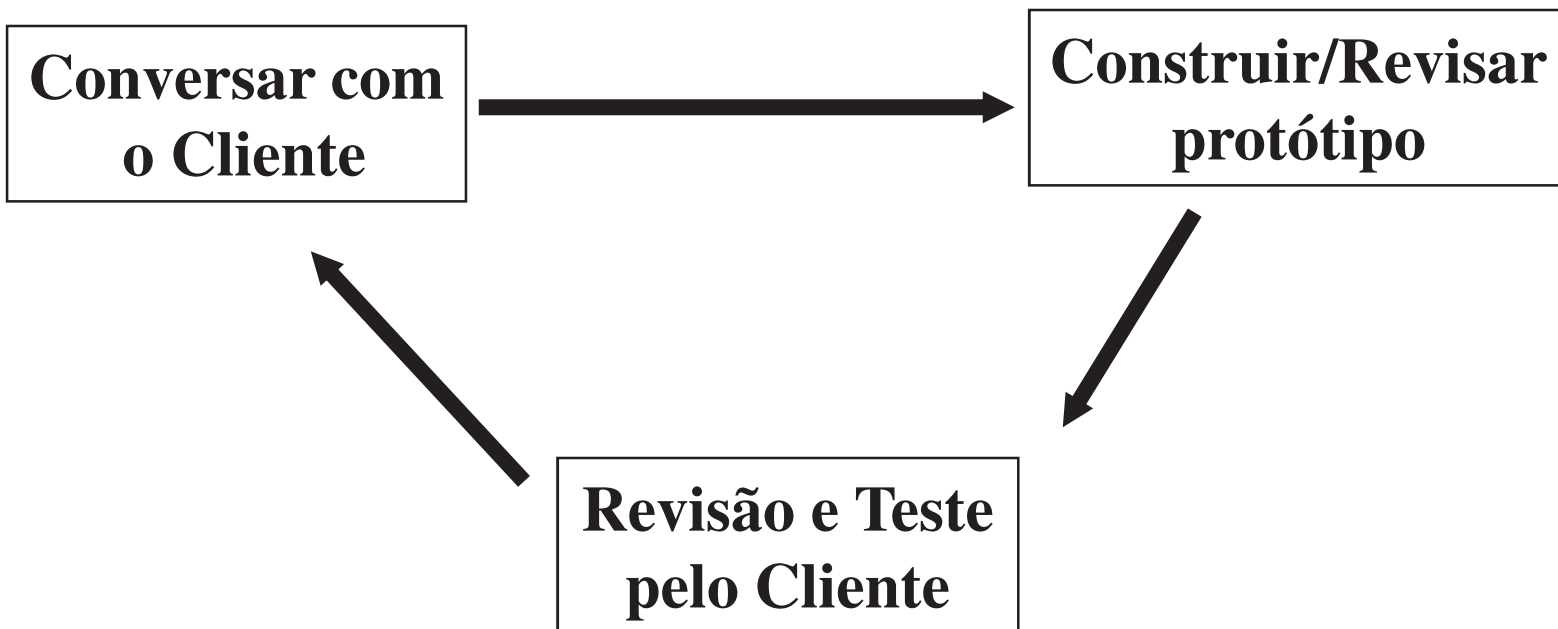
# Modelo Evolucionário

---



# PROTOTIPAÇÃO

---



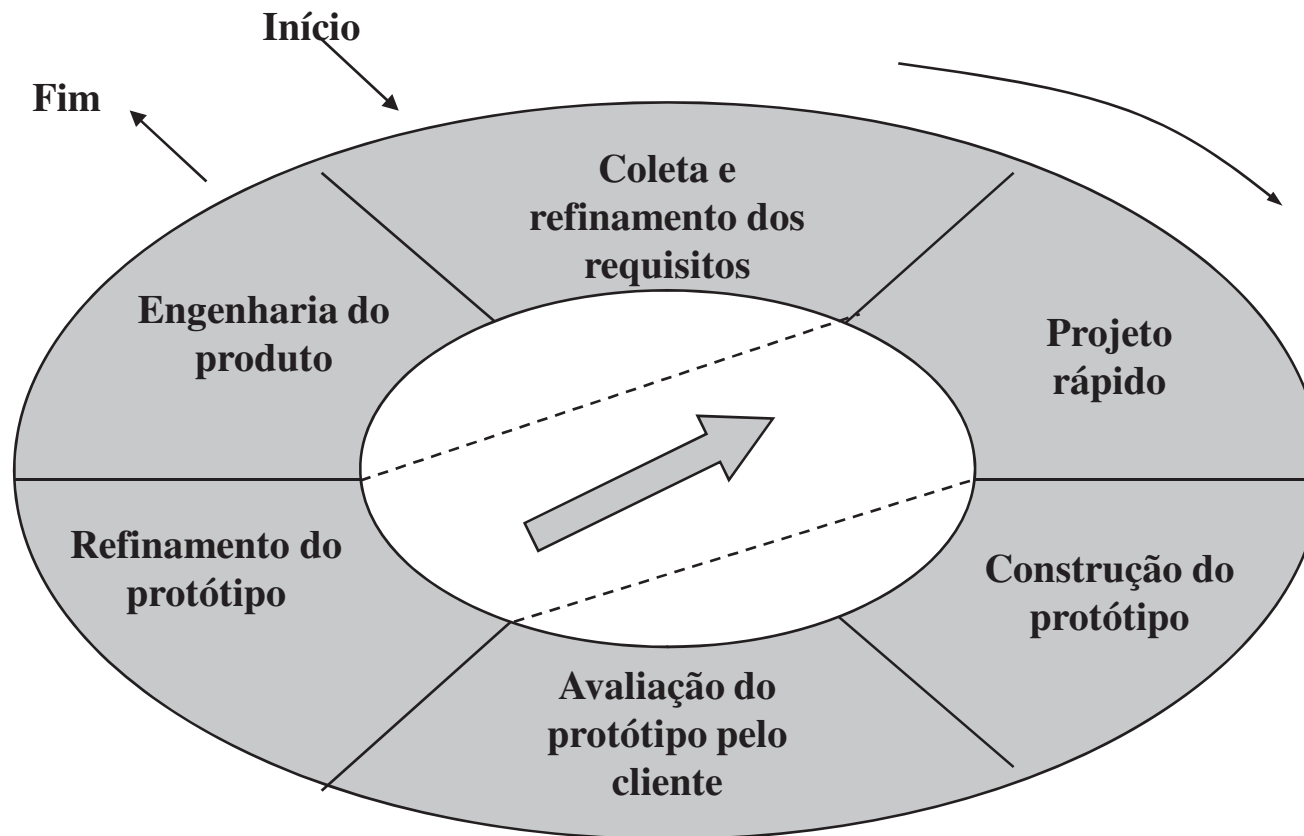
# PROTOTIPAÇÃO

---

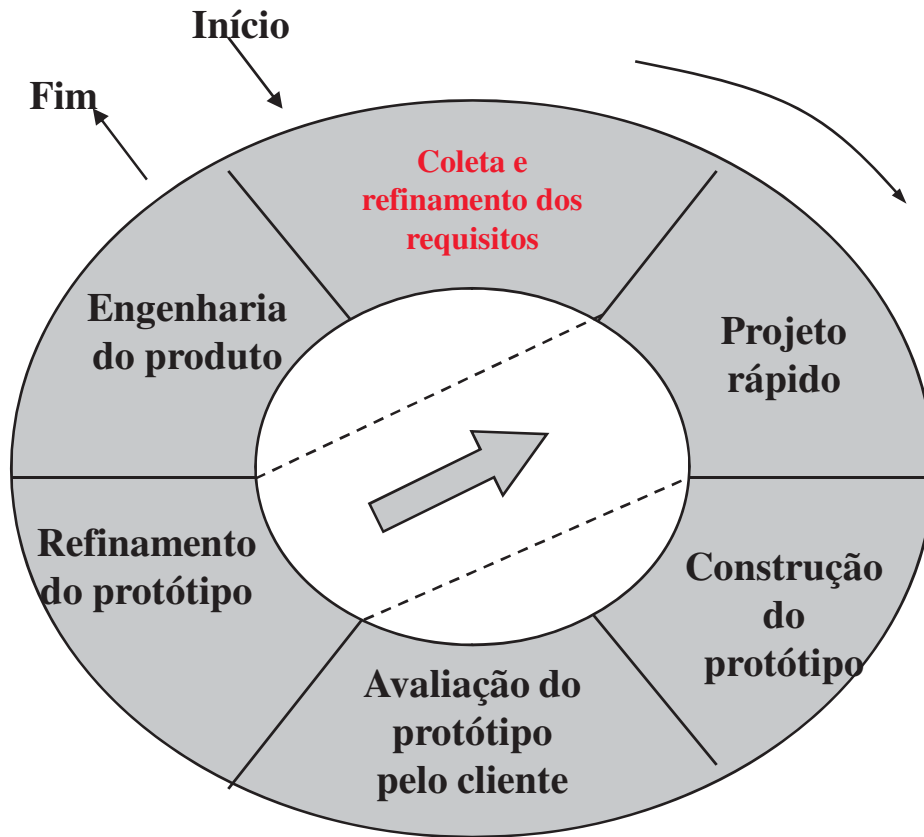
- Obter os requisitos do cliente e, a partir disso, desenvolver uma melhor definição de requisitos para o sistema.
- Concentra em fazer experimentos com partes dos requisitos que estejam mal entendidos.
- Usuário define uma série de objetos para o produto de software, mas não consegue identificar detalhes de entrada, processamento ou requisitos de saída.
- O desenvolvedor está incerto sobre a eficiência de um algoritmo, a adaptação de um sistema operacional ou ainda sobre a forma da interação homem-máquina.

# PROTOTIPAÇÃO

---



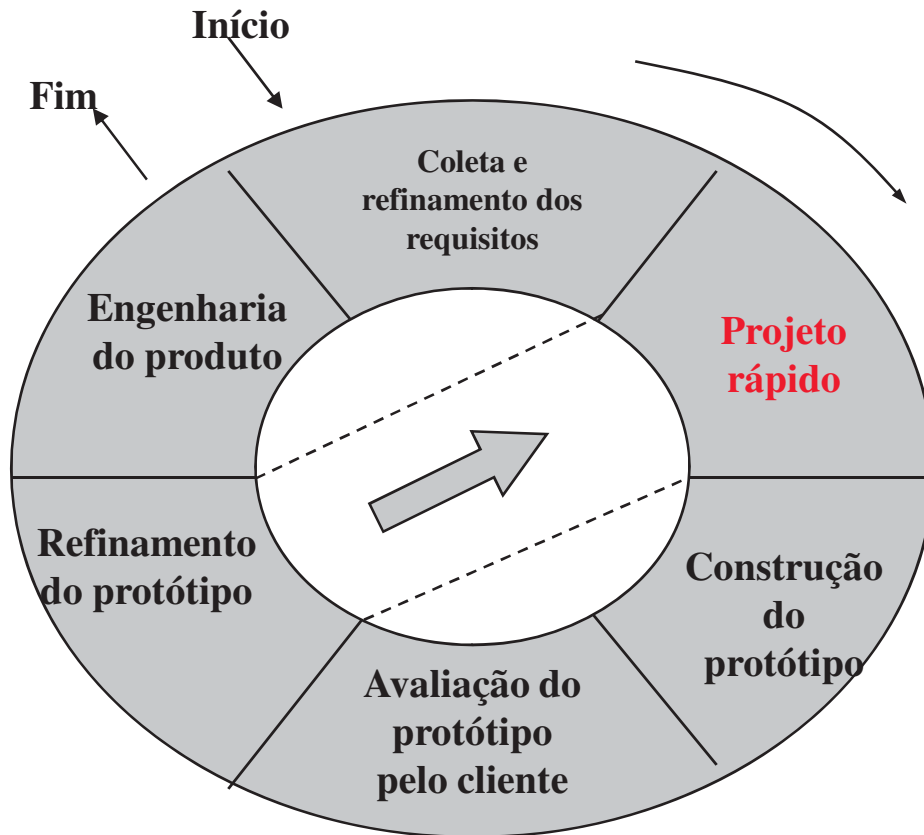
# PROTOTIPAÇÃO



## **COLETA DOS REQUISITOS:**

desenvolvedor e cliente definem os objetivos gerais do software, identificam quais requisitos são conhecidos e as áreas que necessitam de definições adicionais

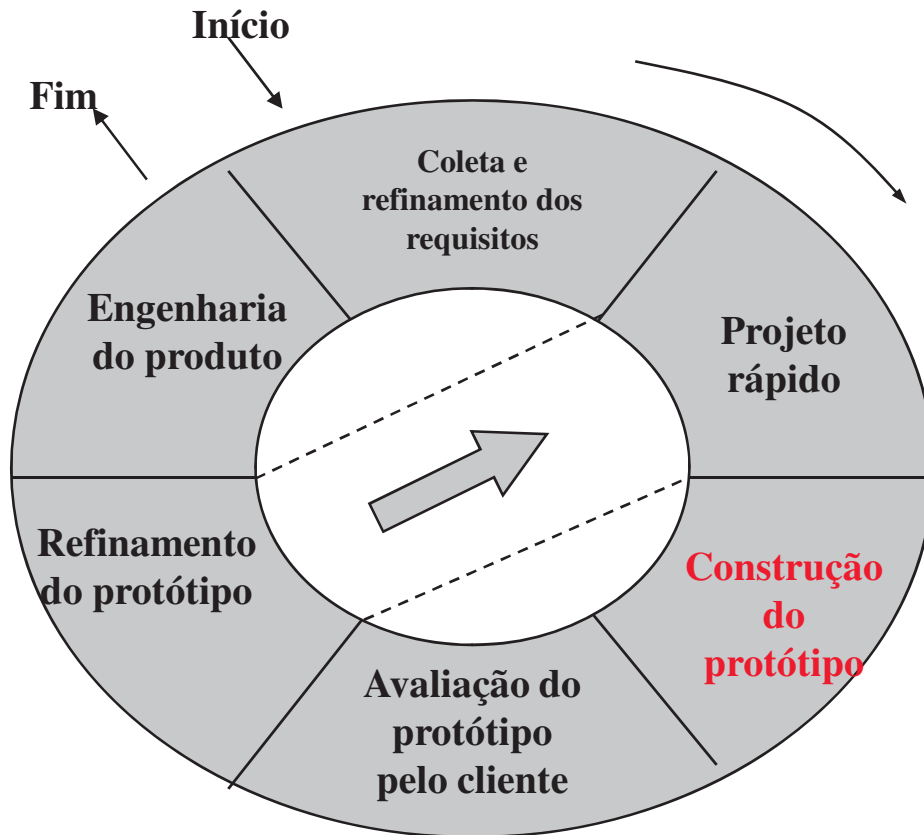
# PROTOTIPAÇÃO



## **PROJETO RÁPIDO:**

representação dos aspectos do software que são visíveis ao usuário (abordagens de entrada e formatos de saída)

# PROTOTIPAÇÃO

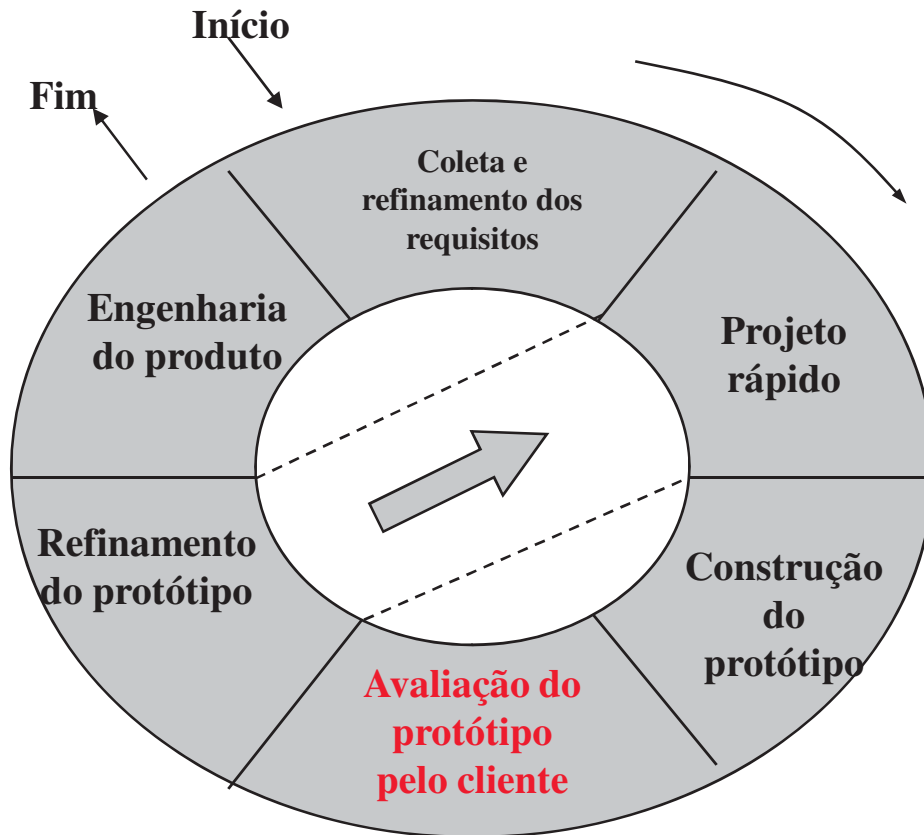


## CONSTRUÇÃO PROTÓTIPO:

Implementação do projeto rápido serve como o “primeiro sistema” - recomendado que se jogue fora futuramente



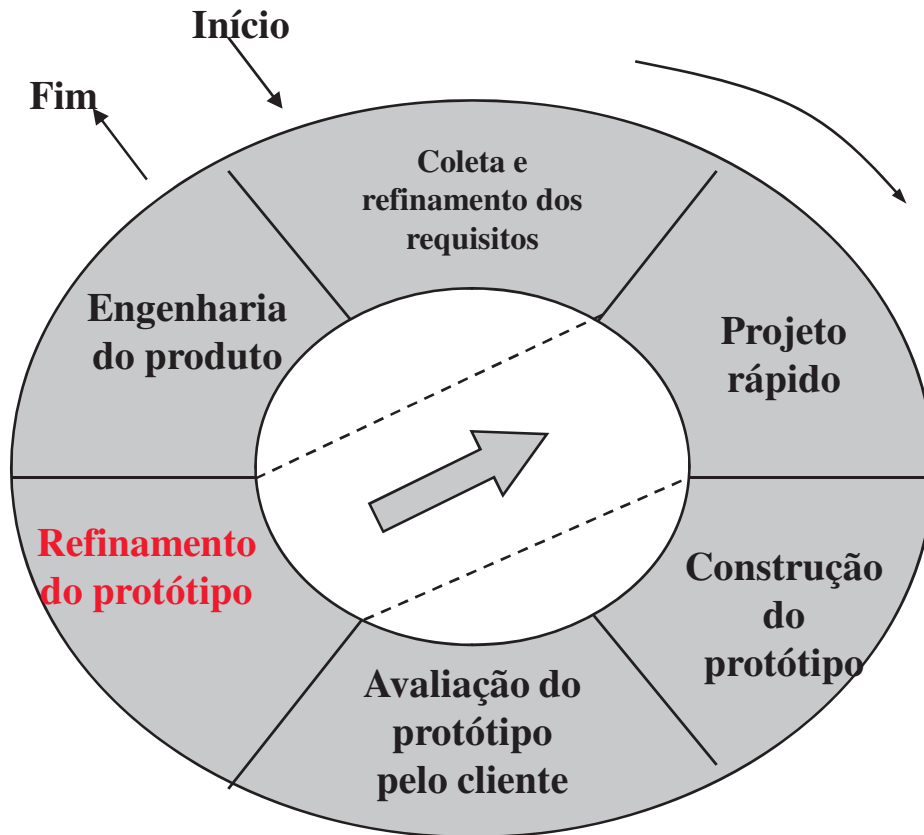
# PROTOTIPAÇÃO



## AVALIAÇÃO DO PROTÓTIPO:

Cliente e desenvolvedor  
avaliam o protótipo

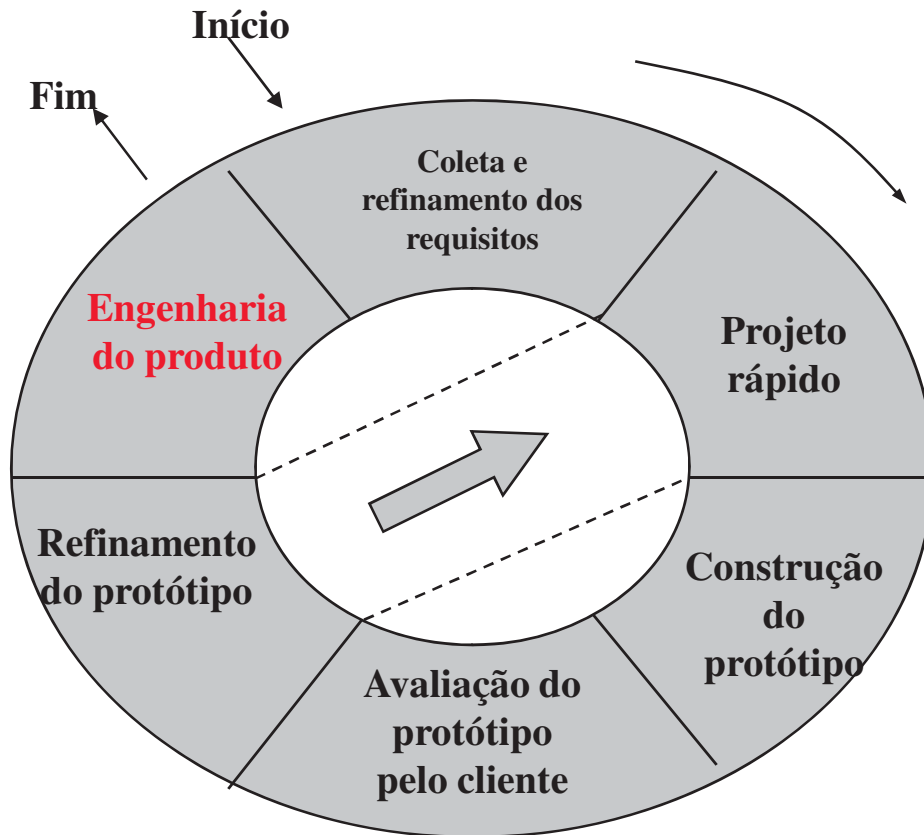
# PROTOTIPAÇÃO



## REFINAMENTO DOS REQUISITOS:

Cliente e desenvolvedor refinam os requisitos do software a ser desenvolvido.

# PROTOTIPAÇÃO



## CONSTRUÇÃO

### PRODUTO:

identificados os requisitos, o protótipo deve ser descartado e a versão de produção deve ser construída considerando os critérios de qualidade.

# Problemas com a Prototipação

---

- O processo não é visível.
- Os sistemas são freqüentemente mal-estruturados e mal-documentados.
- Pode exigir ferramentas e técnicas especiais.
- Processo não é claro, dificuldade de planejamento e gerenciamento.

# Problemas com a Prototipação

---

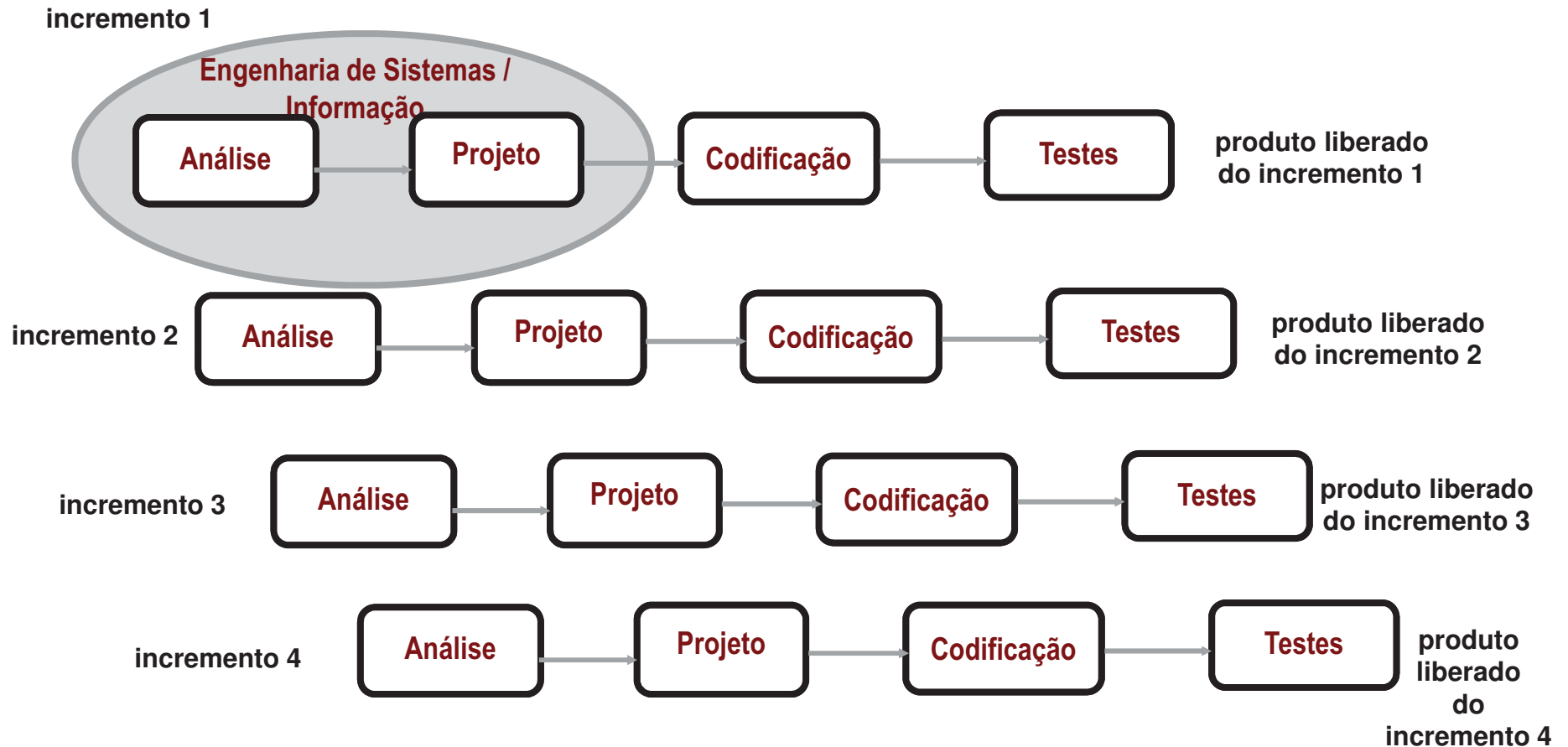
- Desenvolvedor frequentemente faz uma implementação comprometida (utilizando o que está disponível) com o objetivo de produzir rapidamente um protótipo. Depois de um tempo ele se familiariza com essas escolhas, e esquece que elas não são apropriadas para o produto final.
- Cliente não sabe que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a manutenibilidade a longo prazo.

# Modelo Incremental

---

- Combina elementos do Modelo Linear com a filosofia da Prototipação.
- Aplica sequências lineares numa abordagem de “saltos” à medida que o tempo progride.
- Cada sequência linear produz um incremento do software (proc. de texto)
- O processo se repete até que um produto completo seja produzido.
- Difere da Prototipação, pois a cada incremento produz uma versão operacional do software.

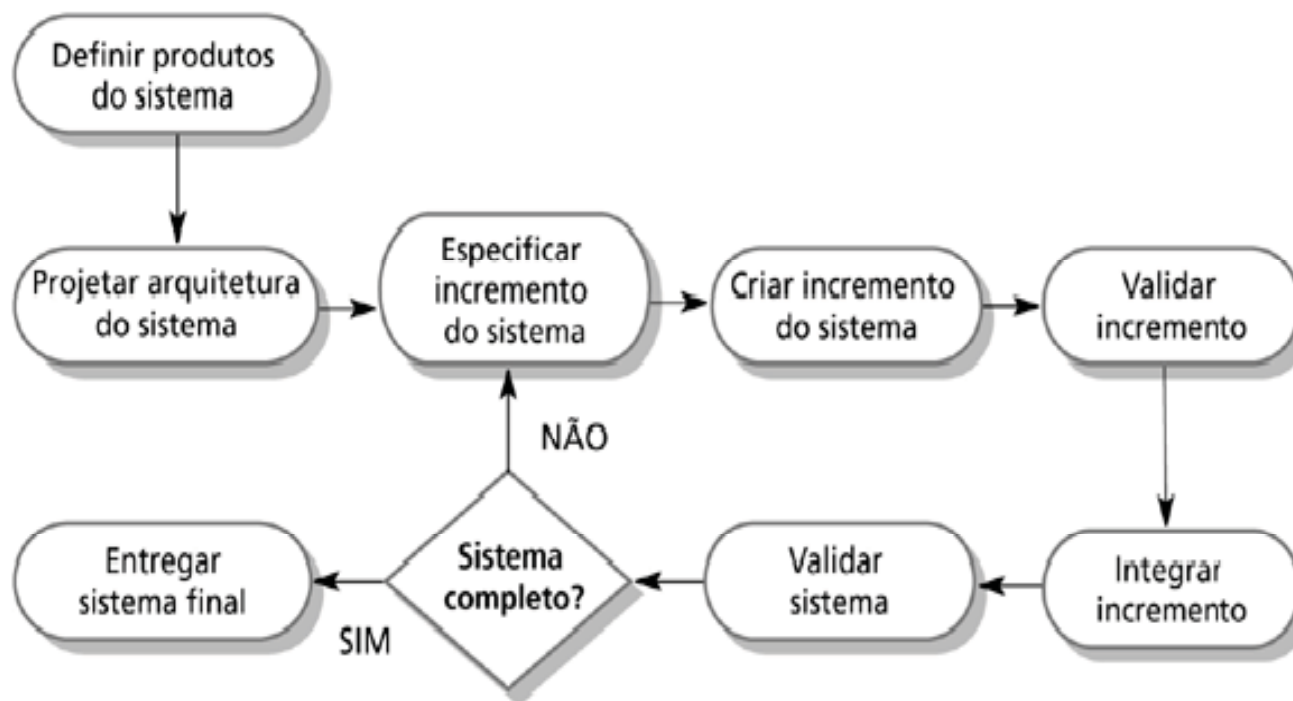
# Modelo Incremental



# Um processo de desenvolvimento iterativo

**Figura 17.1**

Processo de desenvolvimento iterativo.





# Vantagens do desenvolvimento incremental

---

- **Entrega acelerada dos serviços de cliente.**

Cada incremento fornece a funcionalidade de mais alta prioridade para o cliente.

- **Engajamento do usuário com o sistema.**

Os usuários têm de estar envolvidos no processo de desenvolvimento, o que significa que o sistema muito provavelmente atenderá aos seus requisitos, e que os usuários estarão mais comprometidos com ele.

# Problemas com desenvolvimento incremental

---

- **Problemas de gerenciamento**

O progresso pode ser difícil de julgar e os problemas, difíceis de serem encontrados, porque não há documentação que mostre o que foi feito.

- **Problemas contratuais**

O contrato normal pode incluir uma especificação; sem uma especificação, formulários diferentes de contrato têm de ser usados.

- **Problemas de validação**

Sem uma especificação, contra o que o sistema está sendo testado.

- **Problemas de manutenção**

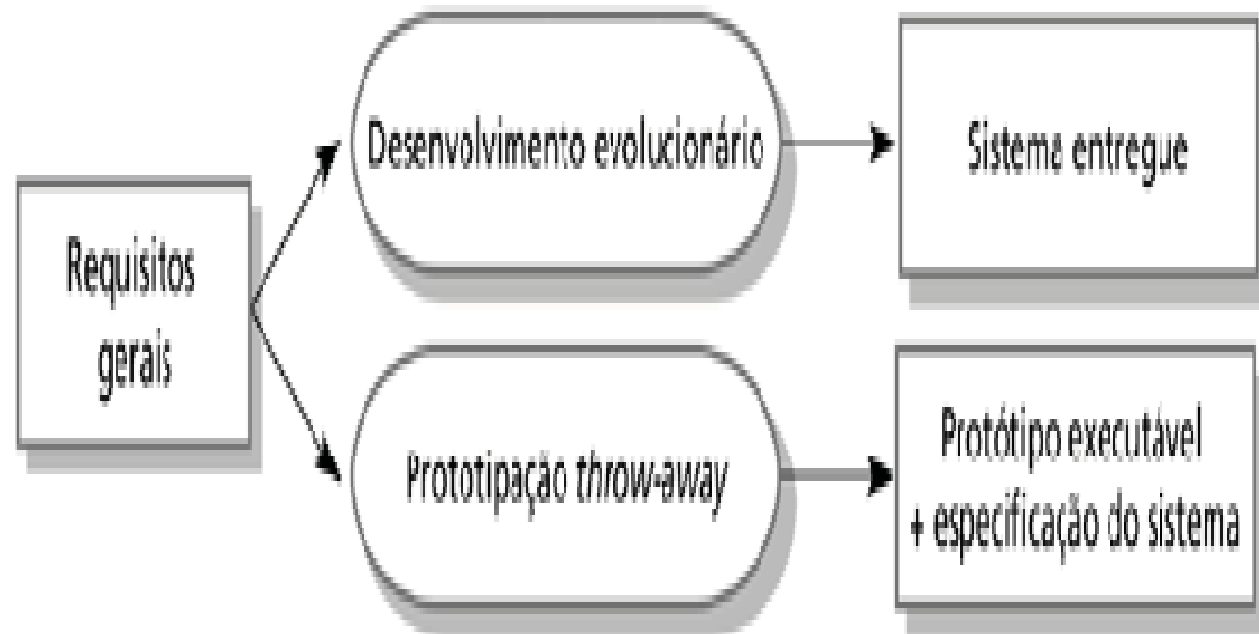
Mudanças contínuas tendem a corromper a estrutura do software, o que torna mais dispendioso mudar e evoluir para atender aos novos requisitos.

# Desenvolvimento incremental e prototipação

---

Figura 17.2

Desenvolvimento incremental e prototipação.



# Desenvolvimento Incremental x Prototipação

---

- Para alguns sistemas grandes, o desenvolvimento e a entrega **incremental e iterativa** pode não ser prático; isso é especialmente verdadeiro quando múltiplas equipes estão trabalhando em localidades diferentes.
- **Prototipação**, no sentido de um sistema experimental, é desenvolvido como base para a formulação de requisitos que podem ser usados. O processo de prototipação inicia com aqueles requisitos que não são bem compreendidos. Esse sistema é descartado quando a especificação do sistema for acordado.

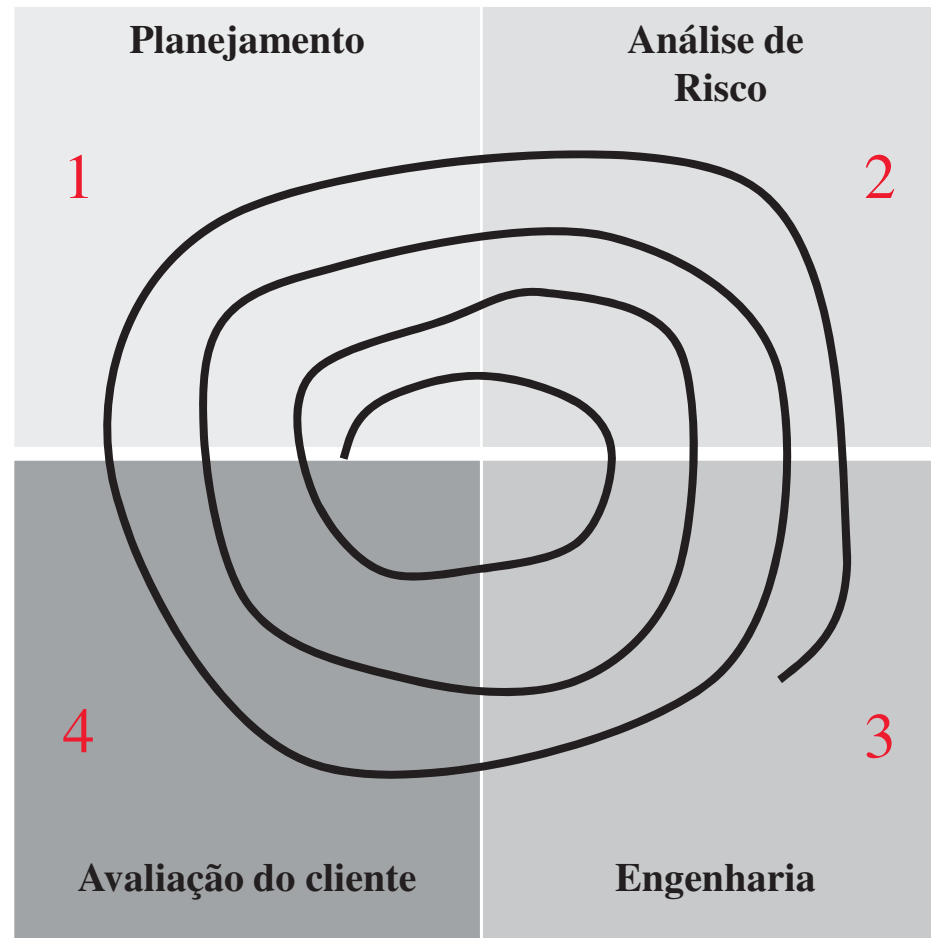
# Modelo Espiral (Boehm)

---

- O processo é representado como uma espiral em lugar de ser representado como uma sequência de atividades.
- Cada loop na espiral representa uma fase do processo de software.
- Capacita o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva.
- Não existem fases fixas.
- Engloba as melhores características do ciclo de vida Clássico como o da **PROTOTIPAÇÃO**, adicionando um novo elemento: a **ANÁLISE DOS RISCOS**.

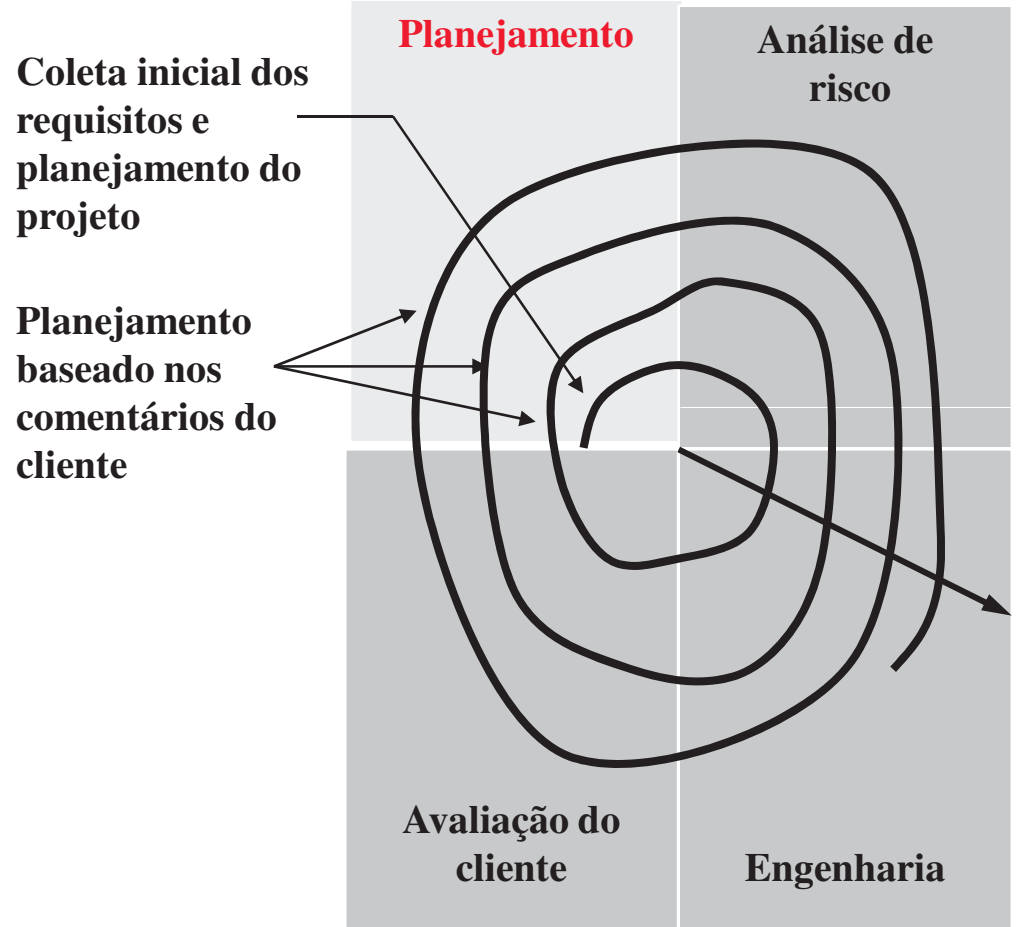
# Modelo Espiral

---

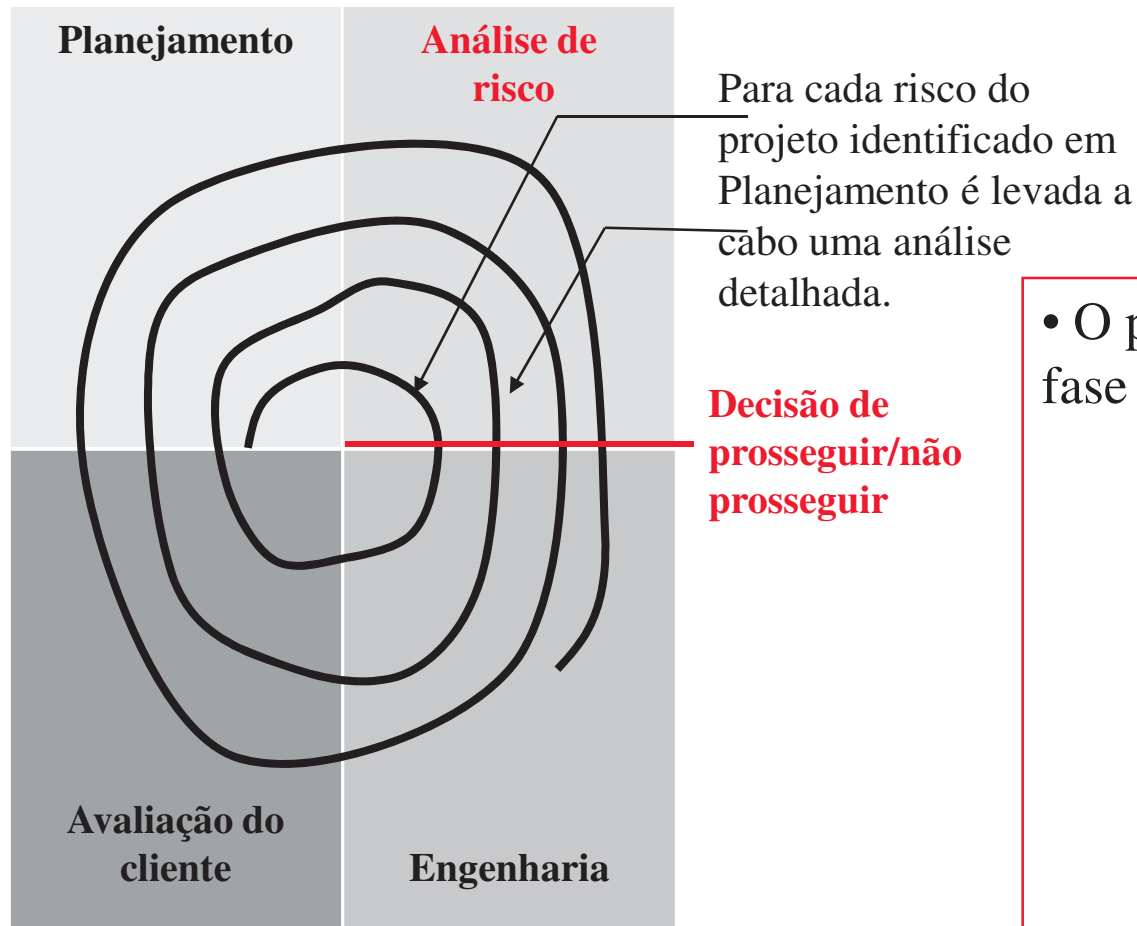


# Modelo Espiral

- São identificados objetivos específicos, tais como desempenho e funcionalidade.
- São determinadas alternativas para atingir estes objetivos.
- São identificadas restrições do processo e do produto e é elaborado um relatório de gestão detalhado.
- Estratégias alternativas, dependendo dos riscos detectados, podem ser planejados.



# Modelo Espiral



- O projeto é revisado e a próxima fase da espiral é planejada.
- Toma-se decisão sobre avançar para mais uma volta na espiral.
- Se for para avançar são desenhados os planos para a próxima fase do projeto.

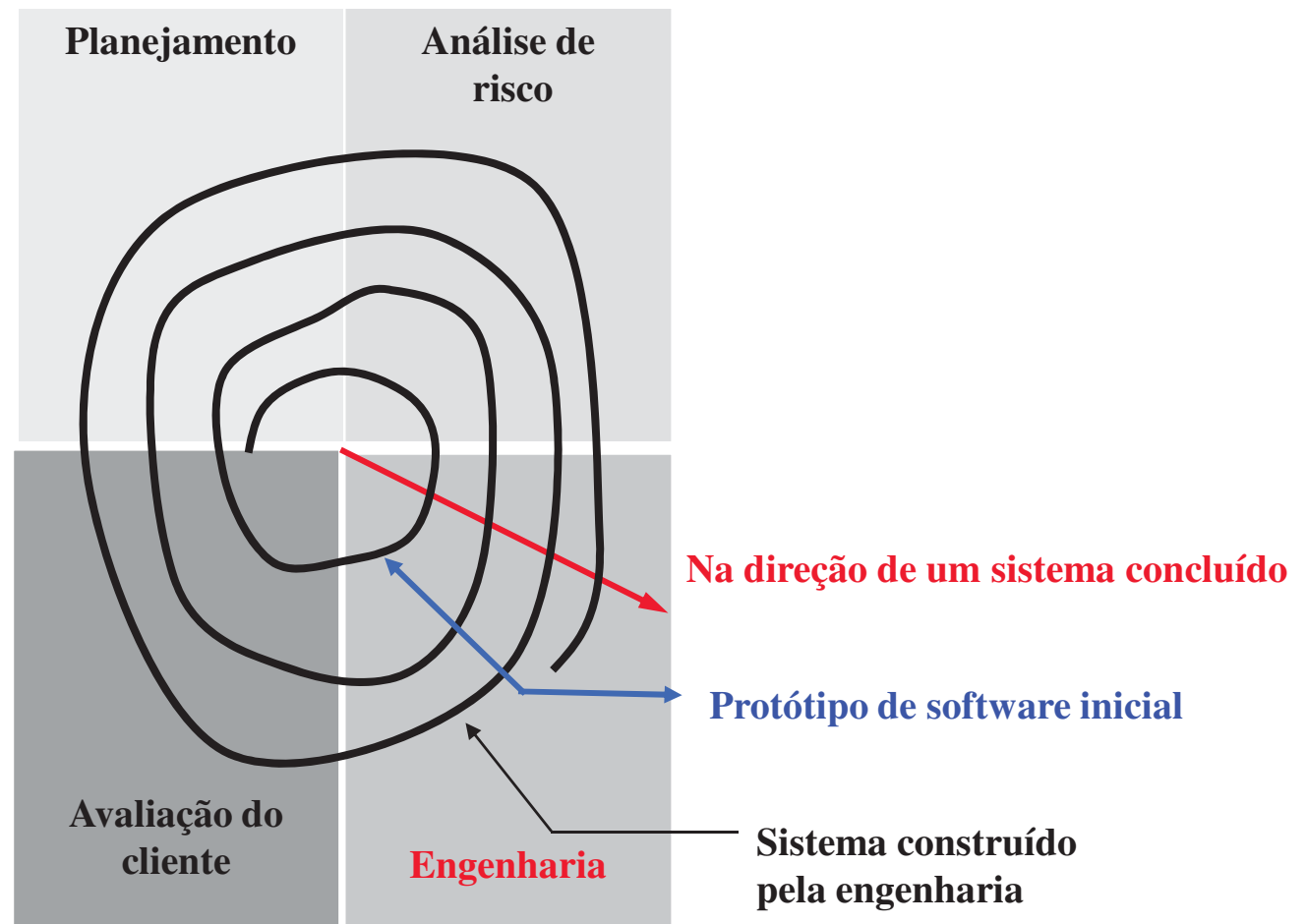


# Modelo Espiral

---

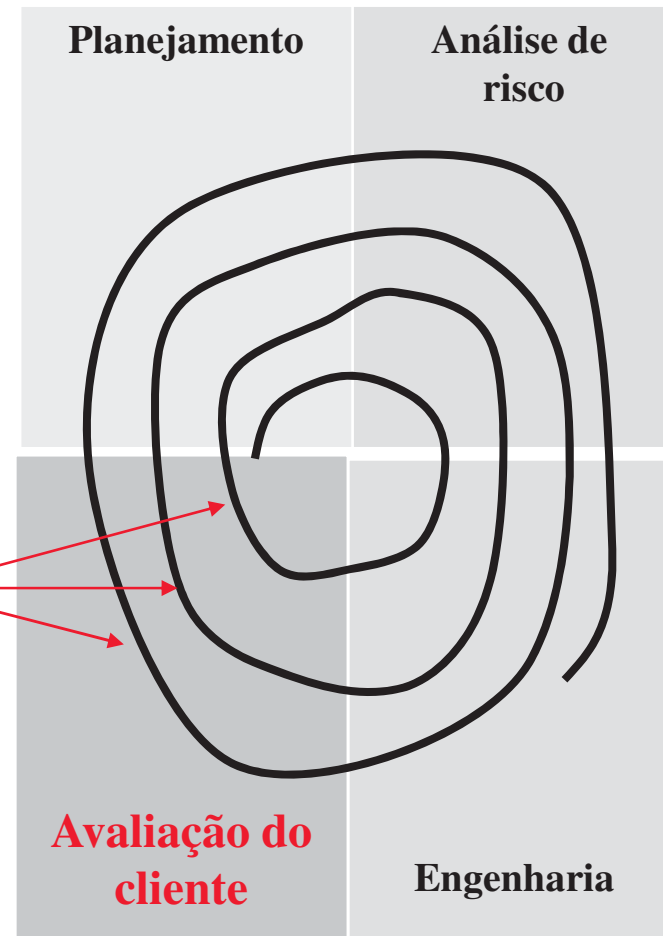
- Escolher um modelo de desenvolvimento para o sistema
  - Riscos significativos na interface com o utilizador → desenvolvimento evolutivo.
  - Riscos de segurança → Desenvolvimento Formal
  - Riscos na integração dos sub-sistemas → Modelo Cascata

# Modelo Espiral



# Modelo Espiral

- tarefas requeridas para obter um feedback do cliente baseado na avaliação da representação do software criado durante a fase de engenharia e implementado durante a fase de instalação



# Desenvolvimento Rápido de Software

---

- Métodos ágeis
- Extreme programming
- Desenvolvimento rápido de aplicações
- Prototipação de software

# Desenvolvimento Rápido de Software

---

- Devido à rápida mudança dos ambientes de negócio, os negócios devem responder às novas oportunidades e à competição.
- Isso requer software e desenvolvimento rápido, e a entrega é, frequentemente, o requisito mais crítico para sistemas de software.
- Os negócios podem estar dispostos a aceitar um software de baixa qualidade se a entrega rápida e a funcionalidade essencial for possível.

# Métodos Ágeis

---

- A insatisfação com os overheads envolvidos nos métodos de projeto levou à criação dos métodos ágeis. Esses métodos:
  - Enfocam o código ao invés do projeto;
  - São baseados na abordagem iterativa para desenvolvimento de software;
  - São destinados a entregar software de trabalho e evoluí-lo rapidamente para atender aos requisitos que se alteram.
- Os métodos ágeis são, provavelmente, os mais adequados para sistemas de negócio de porte pequeno/médio ou produtos para PC.

# Princípios dos Métodos Ágeis

---

<b>Envolvimento do cliente</b>	Clientes devem ser profundamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar as iterações do sistema.
<b>Entrega incremental</b>	O software é desenvolvido em incrementos e o cliente especifica os requisitos a serem incluídos em cada incremento.
<b>Pessoas, não processo</b>	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Os membros da equipe devem desenvolver suas próprias maneiras de trabalhar sem processos prescritivos.
<b>Aceite as mudanças</b>	Projete o sistema para acomodar mudanças.
<b>Mantenha a simplicidade</b>	Elimine a complexidade do sistema.

# Problemas com Métodos Ágeis

---

- Difícil manter o interesse dos clientes que estão envolvidos no processo.
- Os membros da equipe podem ser inadequados para o intenso envolvimento que caracteriza os métodos ágeis.
- A priorização de mudanças pode ser difícil onde existem múltiplos stakeholders.
- A manutenção da simplicidade requer trabalho extra.
- Problemas nos contratos.



# Extreme Programming

---

- É talvez o mais conhecido e mais amplamente usado dos métodos ágeis.
- A extreme programming (XP) leva uma abordagem ‘extrema’ para desenvolvimento iterativo.
- Novas versões podem ser compiladas várias vezes por dia. Os incrementos são entregues para os clientes a cada 2 semanas.
- Todos os testes devem ser realizados para cada nova versão.

# Os 4 valores de XP

---

- Comunicação
- Simplicidade
- Retorno (feedback)
- Coragem

# Extreme Programming

## A quem se destina

---

- Grupos de 2 a 10 programadores
- Projetos de 1 a 36 meses (calendário)
- De 1000 a 250 000 linhas de código
- Papéis:
  - Programadores (foco central)(sem hierarquia)
  - “Treinador” ou “Técnico” (*coach*)
  - “Acompanhador” (*tracker*)
  - Cliente

# Extreme Programming

## “Treinador” ou “Técnico” (*coach*)

---

- Em geral, o mais experiente do grupo. Identifica quem é bom no que. Comunica-se com outros gerentes e diretoria.
- Concentra-se na execução e evolução técnica do projeto.
- Eventualmente faz programação pareada.
- Não desenha arquitetura, apenas chama a atenção para oportunidades de melhorias.
- Seu papel diminui à medida em que o time fica mais maduro.

# Extreme Programming *Tracker* (Acompanhador)

---

- A “consciência” do time.
- Coleta estatísticas sobre o andamento do projeto.  
Alguns exemplos:
  - Número de histórias definidas e implementadas.
  - Número de *unit tests*.
  - Número de testes funcionais definidos e funcionando.
  - Número de classes, métodos, linhas de código
- Mantém histórico do progresso.
- Faz estimativas para o futuro.

# Um Dia na Vida de um Programador XP

---

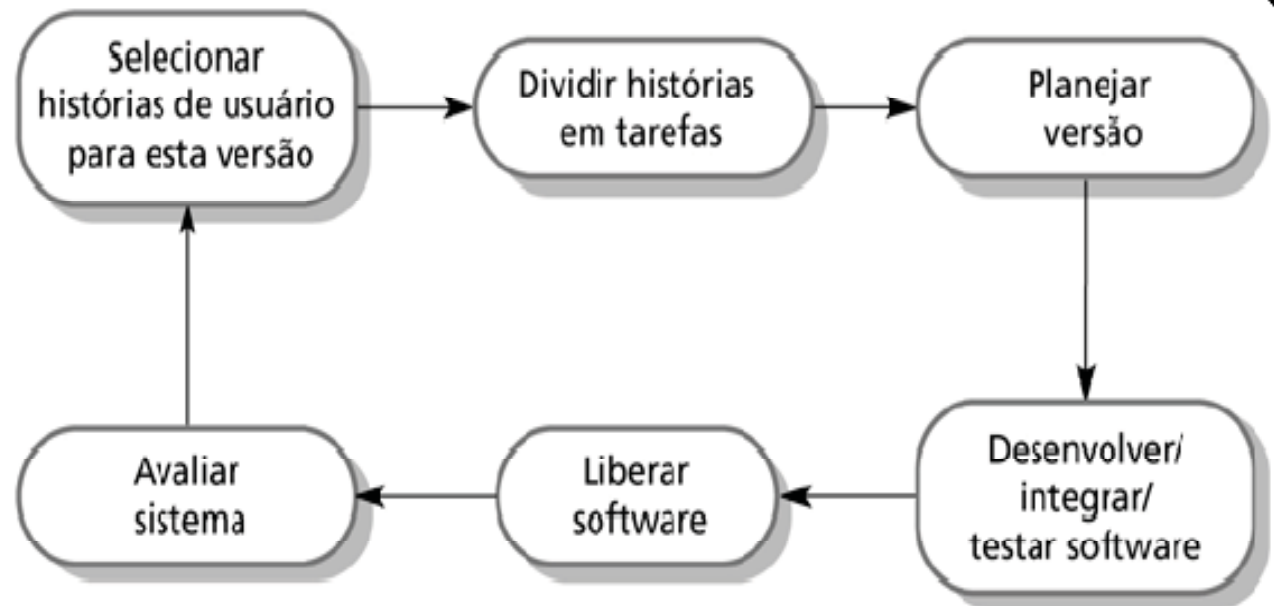
- Escolhe uma história do cliente.
- Procura um par livre.
- Escolhe um computador para programação pareada (*pair programming*).
- Seleciona uma tarefa claramente relacionada a uma característica (*feature*) desejada pelo cliente.

# O ciclo de release em XP

---

**Figura 17.3**

Ciclo de um release em  
*extreme programming*



# Práticas do Extreme Programming

---

<b>Planejamento Incremental</b>	Registrados em cartões de histórias
<b>Pequenos releases</b>	Conjunto mínimo útil de funcionalidade é desenvolvido
<b>Projeto simples</b>	Projeto suficiente para atender aos requisitos atuais.
<b>Desenvolvimento test-first</b>	Uso um framework automatizado.
<b>Refactoring</b>	Espera-se que todos os desenvolvedores recriem o código continuamente.
<b>Programação em pares</b>	Os desenvolvedores trabalham em pares.
<b>Propriedade coletiva</b>	Os pares trabalham em todas as áreas do sistema.
<b>Integração contínua</b>	Tarefa concluída é automaticamente integrada ao sistema.
<b>Ritmo sustentável</b>	Não aceitar grande quantidade de horas extras.
<b>Cliente on-site</b>	Um usuário do sistema deve estar disponível em tempo integral.



# XP e métodos ágeis

---

- O desenvolvimento incremental é apoiado em releases de sistema pequenos e frequentes.
- O envolvimento do cliente significa o seu engajamento em tempo integral com a equipe.
- Pessoas, e não processos na programação em pares, propriedade coletiva e um processo que evita longas horas de trabalho.
- As mudanças são apoiadas em releases de sistema regulares.
- Manutenção da simplicidade por de meio de refactoring constante do código.

# Cenários de requisitos

---

- Em um processo XP, os requisitos de usuários são expressos como cenários ou histórias de usuários.
- Essas histórias são escritas em cartões, e a equipe de desenvolvimento quebra-os em tarefas de implementação. Essas tarefas são as bases das estimativas de cronograma e de custo.
- O cliente escolhe as histórias para inclusão no próximo release, baseado nas suas prioridades e nas estimativas de cronograma.

# Cartão de estória para baixar documentos

---

**Figura 17.4**

Cartão de histórias para baixar documentos.



## Baixando e imprimindo um artigo

Primeiro, você seleciona o artigo que deseja em uma lista. Depois você precisa informar ao sistema como quer pagar pelo artigo — isso pode ser feito por meio de uma assinatura, de uma conta empresarial ou por cartão de crédito.

Após, você obtém do sistema um formulário de direitos autorais para preenchimento. Após enviar o formulário, o artigo desejado é baixado para seu computador.

Em seguida, você escolhe uma impressora para imprimir uma cópia do artigo. Você informa ao sistema que a impressão foi bem-sucedida.

Se o artigo for somente para impressão, você não poderá manter uma versão em PDF, de modo que o artigo será excluído automaticamente de seu computador.

# Programação Pareada

---

- Erro de um detectado imediatamente pelo outro (grande economia de tempo).
- Maior diversidade de idéias, técnicas, algoritmos.
- Enquanto um escreve, o outro pensa em contra exemplos, problemas de eficiência, etc.
- Vergonha de escrever código feio (*gambiarrras*) na frente do seu par.

# Propriedade Coletiva do Código

---

- Padrões de estilo adotados pelo grupo inteiro.
- O mais claro possível.
- XP não se baseia em documentações.
- Comentários sempre que necessários e padronizados.
- O código do XP pertence a todos.
- Todo código é integrado e testado depois de algumas horas, um dia no máximo.

# XP e mudanças

---

- A sabedoria convencional na engenharia de software é projetar para mudança. Vale despende tempo e esforço antecipando mudanças quando isso reduz custos posteriores no ciclo de vida.
- O XP, contudo, mantém que isto não vale a pena quando as mudanças não podem ser confiavelmente previstas.
- Por outro lado, ele propõe melhorias constantes de código (refactoring), para tornar as mudanças mais fáceis quando elas têm de ser implementadas.

# Quando XP Não Deve Ser Experimentada?

---

- Quando o cliente não aceita as regras do jogo.
- Quando o cliente quer uma especificação detalhada do sistema antes de começar.
- Quando os programadores não estão dispostos a seguir (todas) as regras.
- Se (quase) todos os programadores do time não são experientes.

# Quando XP Não Deve Ser Experimentada?

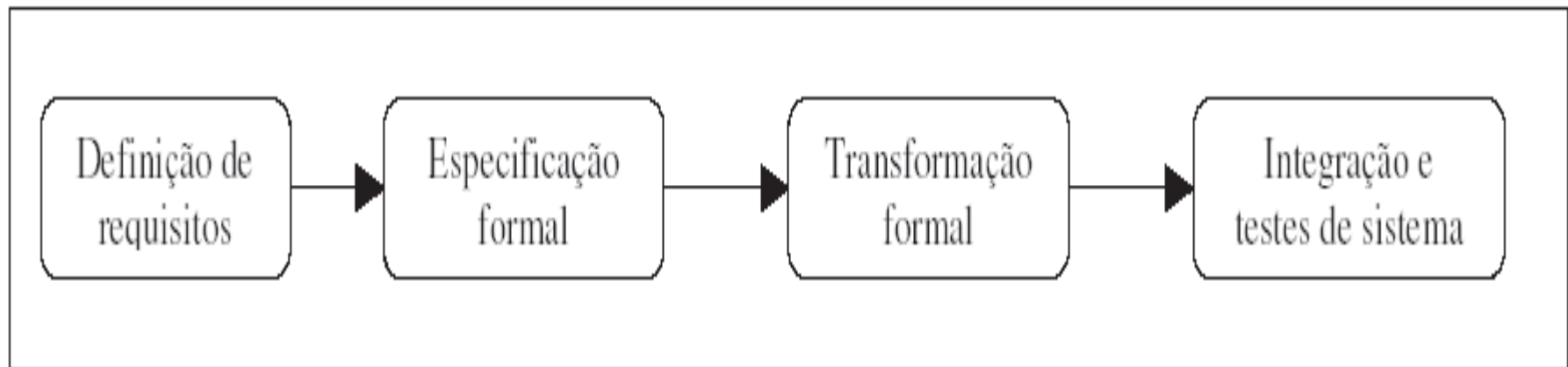
---

- Grupos grandes (>10 programadores).
- Quando *feedback* rápido não é possível:
  - sistema demora 6h para compilar.
  - testes demoram 12h para rodar.
  - exigência de certificação que demora meses.
- Quando o custo de mudanças é essencialmente exponencial.
- Quando não é possível realizar testes (muito raro).



# Modelo de Métodos Formais

---



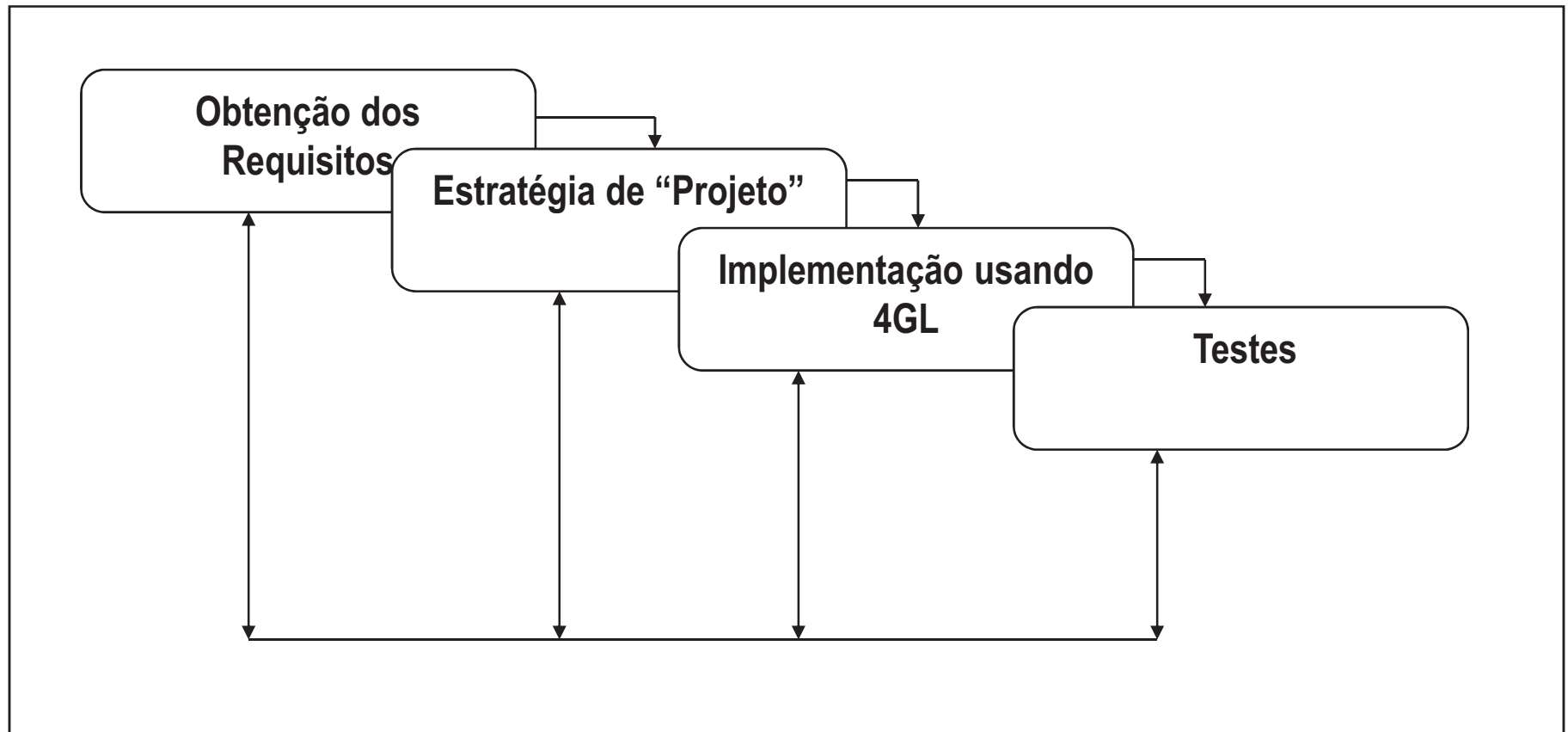
# Modelo de Métodos Formais

---

- Compreende um conjunto de atividades que determinam uma especificação matemática para o software.
- A especificação de requisitos de software é redefinida em uma especificação formal detalhada, que é expressa em uma notação matemática.
- Utilizando métodos formais eliminam-se muitos problemas encontrados nos outros modelos, como p.ex., ambiguidade, incompletitude e inconsistência, que podem ser corrigidas mais facilmente de forma não *ad hoc* mas através de análise matemática.
- Promete o desenvolvimento de software livre de defeitos.

# Técnicas de 4ª Geração

---



# Técnicas de 4ª Geração

---

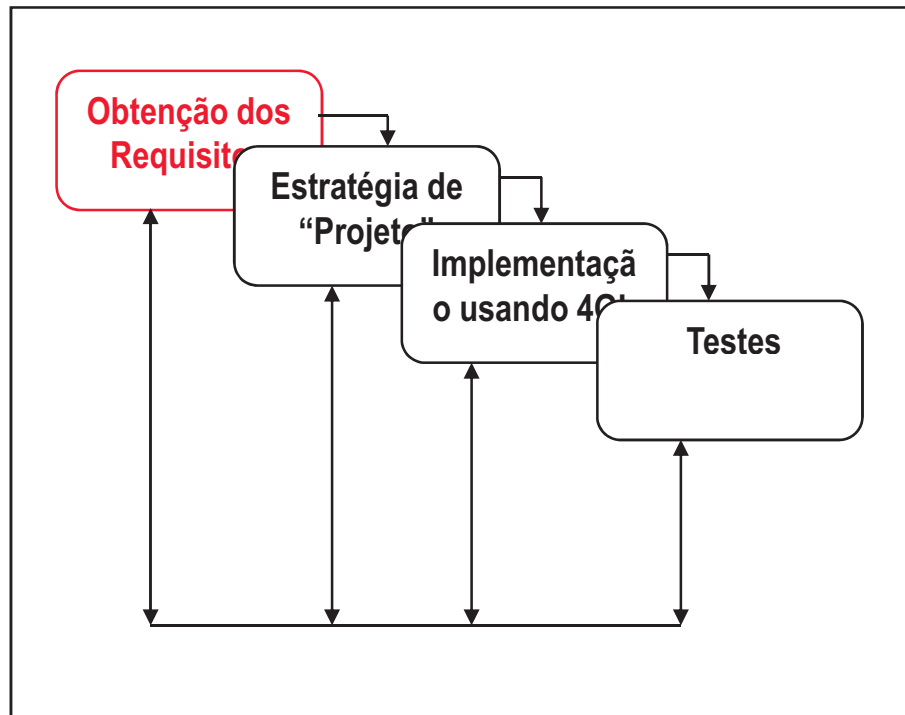
- Concentra-se na capacidade de se especificar o software a uma máquina em um nível que esteja próximo à linguagem natural.
- Engloba um conjunto de ferramentas de software que possibilitam que:
  - o sistema seja especificado em uma linguagem de alto nível e
  - o código fonte seja gerado automaticamente a partir dessas especificações.

# Técnicas de 4ª Geração

---

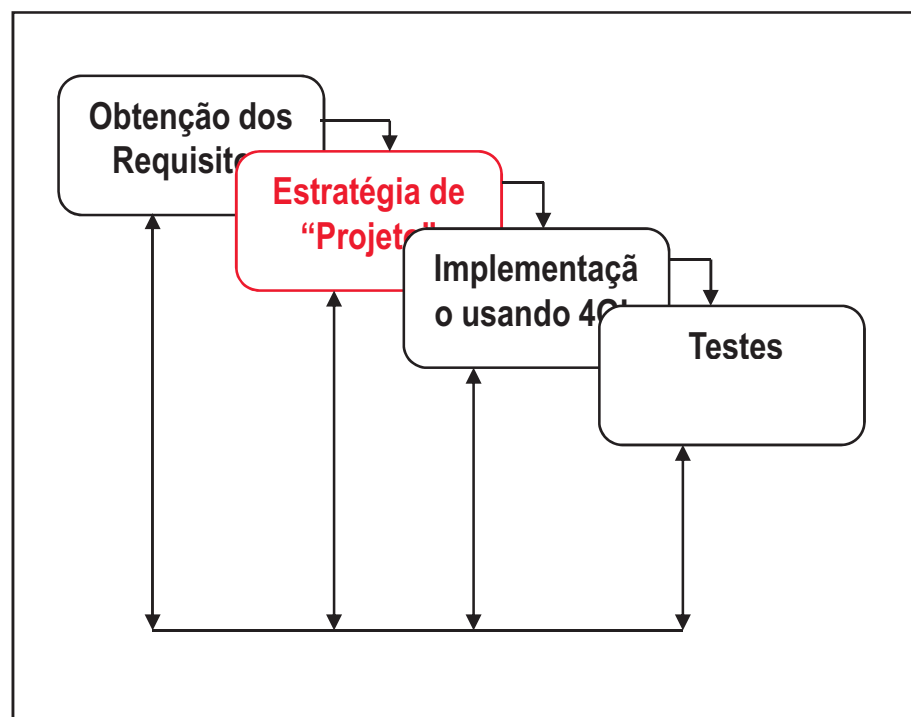
- O ambiente de desenvolvimento inclui as ferramentas:
  - linguagens não procedimentais para consulta de banco de dados
  - geração de relatórios
  - manipulação de dados
  - interação e definição de telas
  - geração de códigos
  - capacidade gráfica de alto nível
  - capacidade de planilhas eletrônicas

# Atividades das Técnicas de 4ª Geração



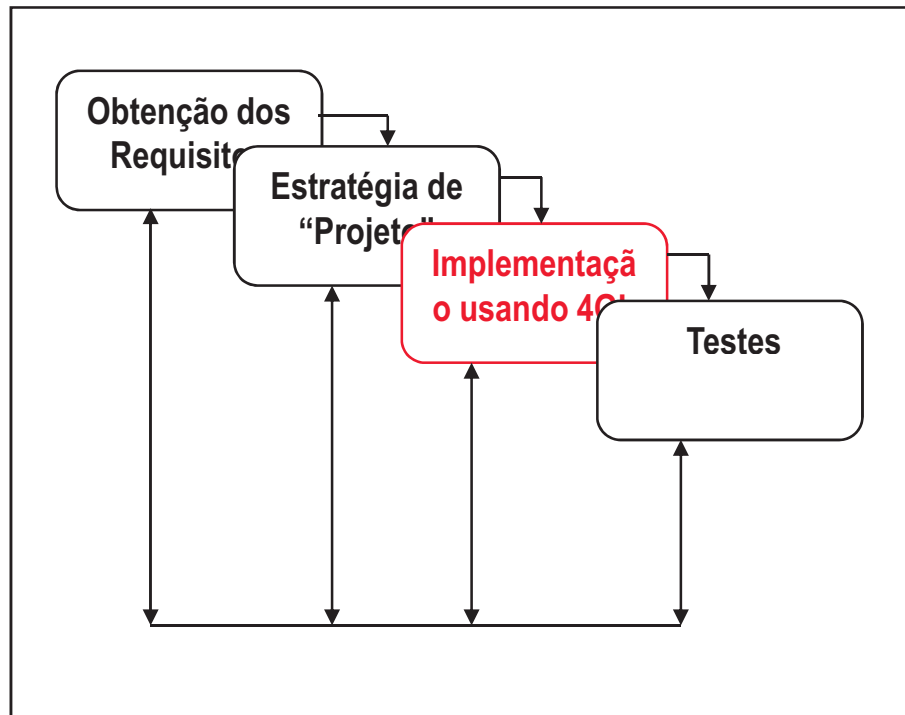
- **OBTENÇÃO DOS REQUISITOS:** o cliente descreve os requisitos os quais são traduzidos para um protótipo operacional
  - O cliente pode estar inseguro quanto aos requisitos
  - O cliente pode ser incapaz de especificar as informações de um modo que uma ferramenta 4GL possa consumir
  - As 4GLs atuais não são sofisticadas suficientemente para acomodar a verdadeira "linguagem natural".

# Atividades das Técnicas de 4ª Geração



- **ESTRATÉGIA DE "PROJETO":**  
para pequenas aplicações é possível mover-se do passo de Obtenção dos Requisitos para o passo de Implementação
  - Para grandes projetos é necessário desenvolver uma estratégia de projeto. De outro modo ocorrerão os mesmos problemas encontrados quando se usa abordagem convencional (baixa qualidade, manutenibilidade ruim, má aceitação do cliente)

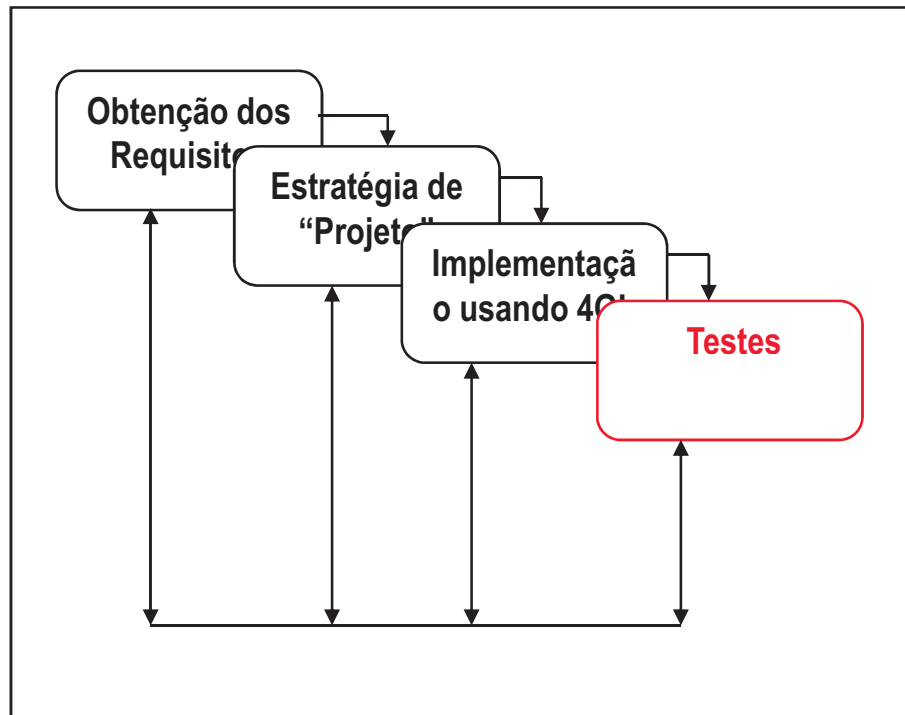
# Atividades das Técnicas de 4ª Geração



- **IMPLEMENTAÇÃO USANDO 4GL:**
  - os resultados desejados são representados de modo que haja geração automática de código.



# Atividades das Técnicas de 4ª Geração



- **TESTE:**
  - o desenvolvedor deve efetuar testes e desenvolver uma documentação significativa.
  - O software desenvolvido deve ser construído de maneira que a manutenção possa ser efetuada prontamente.

# Combinando Paradigmas

